Stage3D API for Java3D

Authors Takao Sumitomo

Stage3D API for Java3D

目次

Preface....3
１：設計思想と概念....4
1.1:Stage3D とは....4
1.2:設計思想....4
1.3.:Stage と Actor....4
1.4:Stage 内の時間について....4
1.5:Acting....5
例 1.1）動作例....6
例 1.2）やってはいけない例....6
1.6:割り込み....7
２：パッケージと主要なクラス....8
2.1:パッケージ....8
2.2:クラス Stage....8
2.3:クラス Actor....9
2.4:クラス CameraMan....11
2.5:クラス Acting....12
2.6:ActingList クラス....13
2.7:大まかな制作手順....13
3：Acting とコマンド....15
3.1:Acting の生成方法....15
3.2:Acting のタイプ....15
3.3:標準で使用可能な Acting....15
3.4:コマンドの作り方....15
3.5:標準 Acting のオプション....16
3.6:コマンドの例....17
3.7:コマンドの作成の方法....18
４：Actor と Acting の作成方法....19
4.1:Actor の作成方法....19
4.2:Acting の作成方法....19
4.3:特殊な例....21
4.4:ノウハウ....22
５：Stage3D.utils パッケージ....23
5.1:SimpleCameraMan....23

Preface

Java3D の登場により Java で 3D を扱うことが可能になった。しかし三次元を扱う以上、複雑なベクトル・マトリクスの演算は必須であるため、プログラムを組む際にそれらも考慮に入れる必要がある。この Scene3D API では人の日常生活で行うようなことを簡単に表現できるようにすることを目的にしている。

この API を使うに当たって必要な知識は以下のようになる。

- Java での基本構文を理解していること
- Java でのオブジェクト指向を理解していること。
- Java3D において三次元オブジェクトの追加程度ができること

このドキュメントでは主に３つのパートに分かれている

- 設計思想と概念
- パッケージと主要なクラス
- Acting とコマンド
- Actor と Acting の作成方法

１：設計思想と概念

1.1:Stage3D とは

Java3D は Java で３次元を扱うための API ですが、そのプログラムの自由度の高さゆえ、簡単なシーンを描写して動かすだけでもそれなりのプログラム量を必要とされます。この Stage3D API は Java3D の細かな制御をすることなくシーンの描写することを目標として作成されました。

1.2:設計思想

Stage3D API は以下のような思想により設計されています。

やりたいことは映画のような一つのシーンを簡単に作れようにすることである。

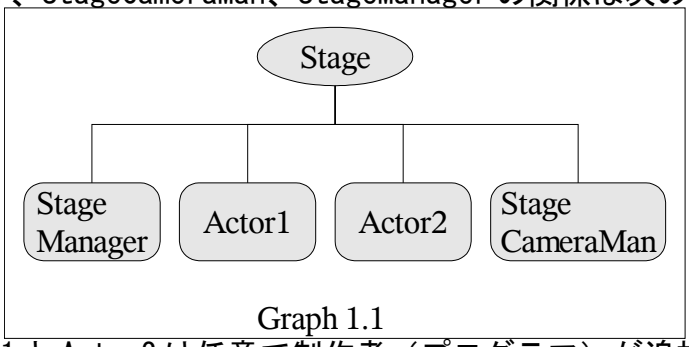
1) シーンを描写するためのステージ（舞台）が必要である。
2) シーンのキャストとして舞台主任・カメラマン・役者が必要。
3) 役者は命令によって演技する。
4) プログラムの複雑化をさけるため、役者の演技は役者とは切り離して考えれるようにする。
5) 指示されるわけではないが、周りに合わせた動き（アドリブ）を行えるように役者は周りの状況を把握することができるようにする。

ここで重要となるキーワードは、ステージ・役者・演技の３つであり、これが Stage3D API の基礎となっている。なお、便宜上これ以降ではステージを「Stage」、役者を「Actor」、演技を「Acting」、舞台主任を「StageManager」、カメラマンを「StageCameraMan」と表記するようにします。

この Stage3D API では上を少し発展して、StageManager と StageCameraMan も一つの Actor として扱います。Stage3D では「Stage に Actor を参加させて、参加している Actor に StageManager から演技を指示される」という形をとります。

1.3.:Stage と Actor

Stage、Actor、StageCameraMan、StageManager の関係は次の図のようになります。



Graph 1.1

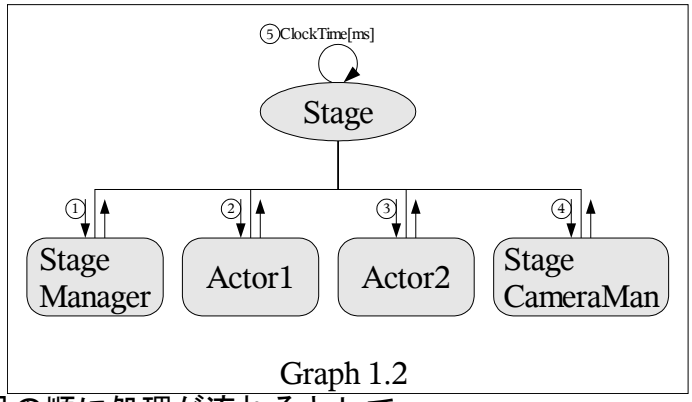
この図で Actor1 と Actor2 は任意で制作者（プログラマ）が追加する物です。StageManager と StageCameraMan は Stage に必ず必要なので常に一人ずついるようになります。このようなとき、それぞれの Actor は「Stage に参加している」、また、Stage は「Actor を持っている」と言うようにします。

Stage にはシーンを作成するための各役者を参加させたり、役者の演技を指示するという命令が必要です。この命令は直接役者に出す場合もあり、Stage の管理人である StageManager を通して命令を出す場合があります。

1.4:Stage 内の時間について

ここでは Stage 内での時間の取り扱いを説明します。ここは Stage3D での時間の扱い方の説明になります。Stage は一定周期（ClockTime、標準は 100ms）ごとにそれぞれの Actor に動作を許すチャンスを与えます。ここではこのチャンスを「ターン」と呼びます。ターンが与えられればその Actor は動作することができ、動作が完了したとき「ターンを終了する」といいます。ターンを貰った Actor は、指示さ

れた自分がそのターンに行うべき動作を行った後、ターンを終了します。そして、一定周期が過ぎるとStageからまたターンを渡されます。図にすると次のようになります。



Graph 1.2

この例では番号の順に処理が流れるとして、
StageManage→Actor1→Actor2→StageCameraManの順にターンが渡され、最後にStageがClockTime[ms]だけ処理を停止したのちに、再度順番にターンを渡していくという動作になります。これから、Stage全体は一定周期ごとに更新されているということになります。たとえば、あるActorがターンを貰ったときに0.1m動いてターンを終了するとすると、ClockTimeが100msのときは一秒間に1m移動するということになります。

Stageと Actorの処理を簡単化すると次のようになります。

１：Stageは一定時間待つ
２：Actorすべてに順番（同時にではない）にターンを渡す。
３：１へ戻る

1.5:Acting

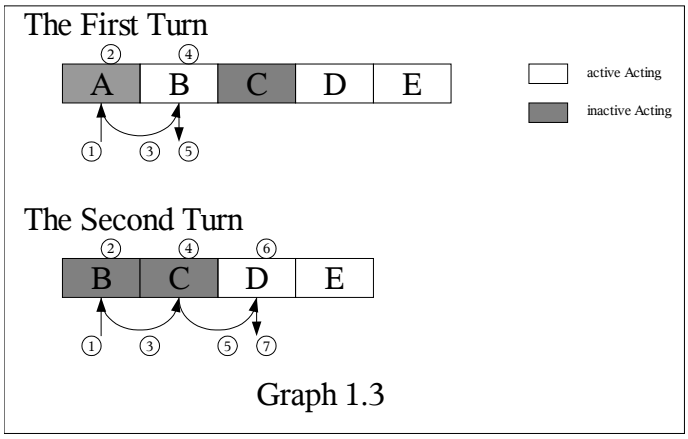
Actorの動作を示した物である。ここでいうActingは「～～まで歩け」や「～～方向を向け」という物のことです。また、ActingはListによって並べられており、このListをActingのListなのでそのままActingListと定義します。Actingは実行することができ、このとき実行することは「Actorがターンを渡されたときに行うべき動作」とする。たとえば「～～まで歩け」というActingは「ターンを渡されるごとに一歩進む」と言うような物である。

Actingは、その演技が終了したかどうかと言うものの判断にactiveかinactiveというフラグを使います。activeには実行可能という意味を、inactiveは実行不可能・または終了と言う意味を持たせています。Actorがターンを渡されたとき、ActingListの前から一番最初のactiveなActingを実行します。このとき、inactiveが設定されたActingは無視されActingListから破棄されます。またActingは実行した後にターンを終了するかどうかを決めます。ターンを終了しない場合は再度ActingListの前から一番最初のactiveなActingを取得して実行します。この動作を「ActingListを実行する」、と言うようにします。

Actingと ActingListの定義は以下のようになります。

Actorは ActingListを持っている。
ActingListは Actingのリストであり、先に入れたものほど前にあるという。
ActingListの最も前にいるactiveなActingを実行中のActingという。
実行中のActingがinactiveになることをActingを終了するという。
ActingListの中のinactiveなActingは無視され破棄される。
Actorはターンを貰ったときにActingListの実行中のActingを実行する。
Actingは activeか inactiveのどちらか一方のフラグを持つ
Actingは実行されるとき、自分を activeか inactiveかを設定できる。
Actingは実行されたとき、ターンを終了するかしないかを必ず決定する。
ターンが終了されたとき、次にターンが回ってくるまでActorは何もできない。
ターンが終了されていないとき、再度ActingListの実行中のActingを実行す

る。



Graph 1.3

例 1.1）動作例

次の Graph 1.3 における処理の流れを示す。
下の ABCDE の列は左が前になる ActingList であり、それぞれが Acting である。白の Acting は active な Acting であり、灰色の Acting は inactive な Acting である。矢印は処理の流れであり、それぞれ次のようになる。なお、「Acting:B と Acting:C は一度実行すると inactive になりターンを終了する」という性質を持つとする。

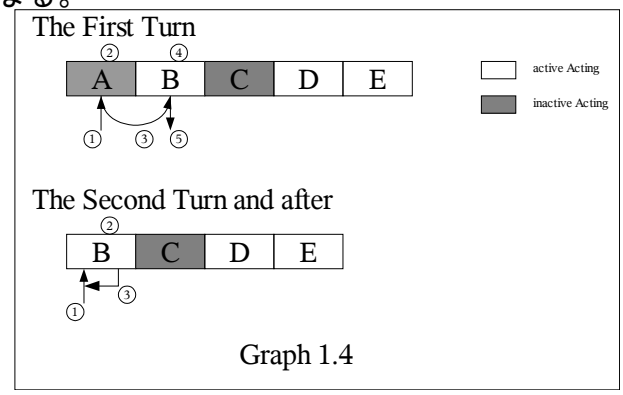
まず第１ターンを見てみる。

1) ターン開始。ActingList の先頭の Acting:A を取得。
2) Acting:A は active でないので破棄（実行されない）。
3) 次の Acting:B を取得。
4) Acting：B は実行され、上の性質より inactive になり、ターンを終了する。
5) ターンを終了する。

第１ターンの終了後２回目のターンでは

1) ターン開始。ActingList の先頭の Acting:B を取得
2) Acting:B は active でないので破棄（実行されない）。
3) 次の Acting:C を取得。
4) Acting:C は active でないので破棄（実行されない）。
5) 次の Acting:C を取得。
6) Acting：C は実行され、上の性質より inactive になり、ターンを終了する。
7) ターンを終了する。

この例では Acting:B と Acting:C は一度実行れたら二度目は実行されないが、３度実行されれば inactive になるという性質を持たせれば３回呼び出された後、inactive になる。



Graph 1.4

例 1.2）やってはいけない例

次の Gpraph 1.4 の例はやってはいけない例である（このような Acting は作成

してはならない）。Acting:Bは例1.1と同じとする。Acting:CはTurnを終了せず、無限回実行してもinactiveにならないActingとする。このときにこの例における処理の流れを説明する。まず一回目のターンは例１.１と同じである。そして２回目のターンは次のようになる。
1) ActingListの先頭のActing:Bを取得
2) Acting：Bは実行され、上の性質よりactiveのまま、ターンを終了しない。
3) ActingListの先頭であるActing:Bを再度実行しようとする（１へ戻る）。
このとき、処理がループしてしまい、他のActorにターンがわたらないどころかプログラム自体が動作しなくなるので注意しなくてはならない。

次にActingは５つに分類できるのでそれぞれを示しておく
1) inactiveで実行されない物
2) activeで実行されるとターンを終了して、有限回実行されるとinactiveなるもの。
3) activeで実行されてもターンを終了せず、有限回実行されるとinactiveなるもの。
4) activeで実行されるとターンを終了して、無限回実行してもinactiveにならないもの。
5) activeで実行されてもターンを終了せず、無限回実行してもinactiveにならないもの。

１は基本的に動作が終了したActingがinactiveになった場合である。２はもっとも一般的なActingである。３はターンを終了せずに自分が終了するまで連続して呼び出されるものである。
４は無限回実行され続ける物である（中断させる方法として「割り込み要求」という方法があり、それは次で説明する）。５は例1.2で示したやってはいけない例である。

1.6:割り込み

上で「activeで実行されてもターンを終了せず、無限回実行てもinactiveにならないもの」をやってはいけない例としてあげた。次に「activeで実行されるとターンを終了して、無限回実行てもinactiveにならないもの」を考える。このようなActingが存在しているとどうなるか考えてみよう。このようなActingが存在しているとき、そのActorはそのActingを実行するだけで他のActingを実行できなくなる。もっとも始めから他のActingを実行しないつもりならよいが、途中で何かの刺激で中断するようにすることも必要である。たとえば、「誰かがｰするまで待つ」というようなActingには必要である。これを実現するために「割り込み」を導入する。

Actingに外部から変更可能な値を準備しておき、それの値を変更することを「割り込み」または「割り込み要求を出す」と言うようにする。また、その変更可能な値を「シグナル」と呼ぶようにする。Actingは実行されるときにシグナルを調べるようにしておき、割り込み要求によりシグナルが変更されたときに動作を終了する、というようにすれば、外部から（他のActorなど）の刺激により終了することができるActingを作成できる。

２：パッケージと主要なクラス

2.1:パッケージ

Stage3D のパッケージ構造は次のようになる。

```
Stage3D
      |
      +--utils
```

パッケージ Stage3D には基本となるクラスが、パッケージ utils には Stage3D を使ってプログラムを組むときに便利なクラスが納められている。

それぞれのパッケージには以下のようなクラスがある。

・Stage3D

ActAddActor　　ActEnQueue　　ActInterrupt
ActMove　　ActSay　　ActSetAngle
ActSetPosition　　ActThrowEvent　ActTurn
ActWait　　Acting　　ActingList
Actor　　ActorGroup　　CameraMan
Stage

・Stage3D.utils

ActCamSetCamTargetActor　　ActCamSetCamTargetLock
ActCamSetCamTargetPoint　　ActCamSetCamViewActor
ActCamSetCamViewLock　ActCamSetCamViewPoint
SimpleCamActing　　SimpleCameraMan

これを見ると Act~で始まるクラスが多いことに気づくと思います。これらは Acting クラスを継承した物で、すべて動作を表しています（Actor, ActorGroup を除く）。Stage3D クラスの Acting の派生クラスはすべて標準で使用可能になっています。たとえば、ActAddActor は Stage に Actor を追加するための Acting、ActEnQueue は Actor の Queue に Acting を追加する Acting です。このパッケージの基礎となるクラスは Stage,Actor,Acting の三つで、それぞれにいろいろなメソッドがありますが、三次元のシーンを表現するには Acting が使用されます。実際にシーンを作成する方法は３・４章で述べます。

2.2:クラス Stage

Stage クラスは Chapter １の Stage をそのまま Java のクラスにした物である。また、これは同時に Java3D の Locale を継承した物でもある。

このクラスのコンストラクタについて説明する。

```
Stage(VirtualUniverse vu, int clocktime)
```

引数の VurtualUniverse はスーパークラスである Locate クラスのコンストラクタに必要な物である。clocktime はこの Stage のターンの周期を指定する物で単位は ms（ミリ秒）である。

前の章で Stqage には StageManager と StageCameraMan の二つの Actor が必須であると述べた。このコンストラクタでは StageManager は生成され、Stage に所属させます。しかし、カメラマンはこのコンストラクタでは生成も登録もされません。プ

ログラム内で後で生成・登録する必要があります。カメラはActorクラスを継承した別クラスであり、詳細は後述するクラスCameraManで説明します。

このクラスの重要なメソッドについて説明する。

public Actor getActor(String actorname)

Stageに所属するActorで名前がactornameと一致するActorを取得します。

public int getClockTime()

Stageのターンの周期を取得します。これはコンストラクタで指定した物と同じ値を返します。

public void setCameraMan(CameraMan cameraman)

StageにCameraManを設定します。

public void setActorType(String actortype, Class actor)

Stageで使用可能なActorのクラスを指定します。これはActorをStageに参加させるわけではなく、指定されたActorのクラスをStageで使用可能にすることと、その名前をActorのクラスに関係付けることです。

また、ここで指定されたActorのクラスのインスタンスがStage内で生成されるため、このActorのクラスは「public」修飾子が付かない物や、内部クラスは使用できません。

public void setUpwardVec(Vector3d upwardvec)
このStageの上を表すベクトルを設定する。（極座標で角度を得るために必要）
標準では(0,1,0)を使用する。

public void start()

これはRunnableインターフェースのメソッドで、Stage開始を示します。これはすべての設定が終わった後に呼び出すようにしてください。

public void throwEvent(String arg)

何かイベントが起こったときにその内容を引数argとして渡すようにして使用します。この引数argはStageCameraManに渡されます。

ここにStageにActorをStageに参加させるするメソッドは挙がってませんが存在はしています。addActorメソッドで参加させてsetVisibleメソッドで可視にします。しかし、これらは通常使用せずに「Actorを追加する」というActingを使用します。これについてはChapter 3で説明します。

2.3:クラスActor

StageクラスはChapter 1のActorをそのままJavaのクラスにした物である。一つ違う点に、ActorクラスはActingListを複数持っているという点である。これは複数のActingを同時に実行するためのものである。このクラスは三次元オブジェクトの雛形であり、基本的にこのクラスをそのまま使用することはなく、extendsして必要な形にしてから使用する。以下ではextendsされたActorのクラスを「Actorのタイプ」と、Actorのクラスのインスタンスを単に「Actor」と表記する。このクラスは以下のようなプログラム的は目的を持って制作されています。

・Actingを使うことで動作やアニメーションを簡単に使えるようにする。

・Actor同士でお互いの状態を調べられる。

・移動・回転を簡単にするメソッドを持っている。

・プログラマが作成するべき部分を明確にすること

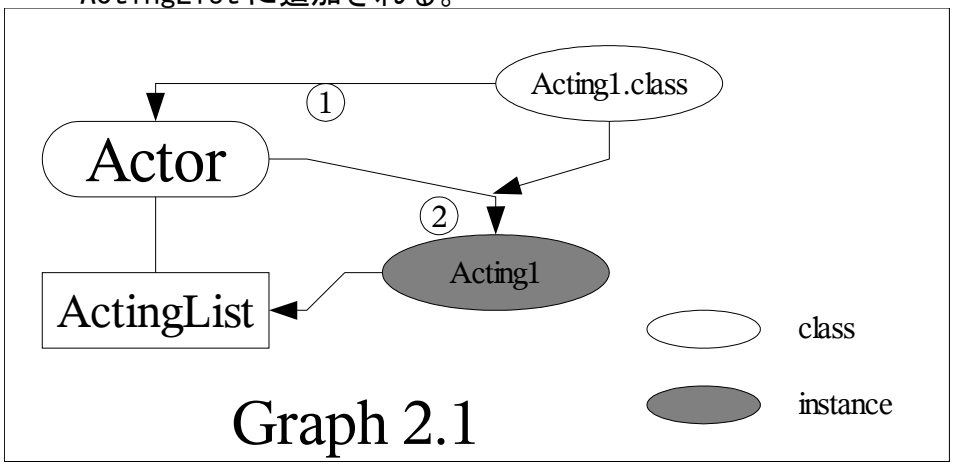
これらをそれぞれのメソッドとあわせて説明します。

・Acting を使うことで動作やアニメーションを簡単に使えるようにする。

Actor のタイプによって使用可能な Acting のクラスが分かれているので、setActing メソッドで Acting のクラスを Actor に登録しない限り、使用可能な Acting は標準で登録されている物だけである。登録された Acting のクラスは getActing メソッドによって Acting のインスタンスが生成可能となり、interpret メソッドにより Acting のインスタンスが生成され、ActingList に追加される。ActingList は ActingList クラスによって実現されている。このクラスについては後述する。また、Actor は ActingList を複数持っているので、複数の Acting を同時に実行することもできる。

Acting を使用する使用する手順は Graph 2.1 にあわせて次のようになるます。

1. コンストラクタ内もしくは getActing メソッドが呼び出される前までに setActing メソッドによって Actor に Acting のクラスを登録する。
2. interpret メソッドにより登録された Acting のインスタンスが生成され ActingList に追加される。



・Actor 同士でお互いの状態を調べられる。

この目的は以下のメソッドによって実現されます。

public Datapack getStatus(int th)
th 番目の Queue の実行中の Acting の状態を返す。存在しない場合は null を返す。
public DataPack getStatus(String actingname)
どれかの Queue の実行中の Acting で名前が actingname と一致する物の状態を返す。存在しない場合は null を返す。
public DataPack[] getStatuses()
それぞれの Queue の始めの Active の Acting の状態を返す。長さは Queue の数（標準は５）だけある。Active な Acting がない Queue の部分は null を返す。

・移動・回転を簡単にするメソッドを持っている。

この目的は以下のメソッドによって実現されます。

public Vector3d getPosition()
現在の位置を返す。
public void movePosition(Vector3d vec)
現在地を vec まで次の ClockTime までに補完しながら移動します。
public void setAngle(Vector3d vec)

角度を latitude（緯度）を vec.x と longitude（経度）を vec.y として指定して、その方向を向かせます。

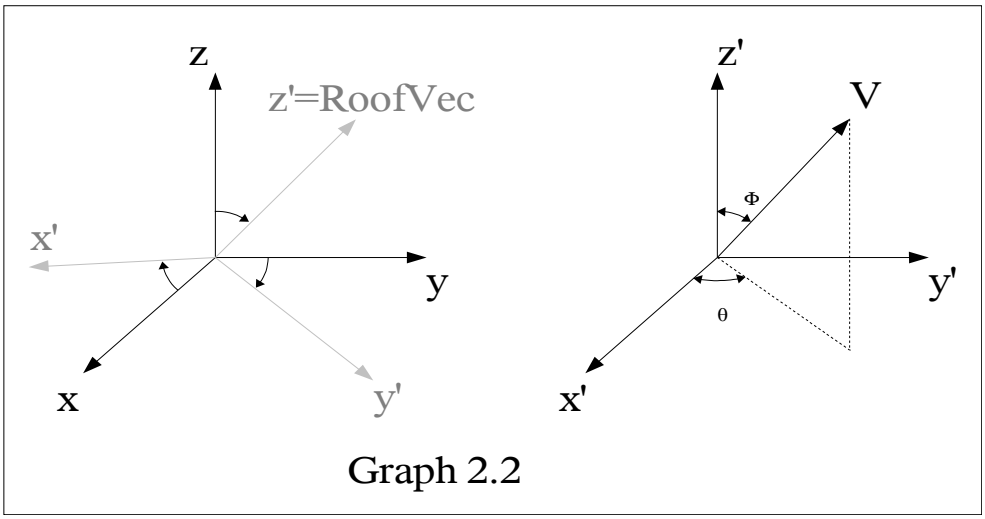
public void setAngleVector(Vector3d vec)

ベクトルが指す方向を向かせる。

public void setPosition(Vector3d vec)

vec を位置ベクトルとしてそこに移動する。

なお setAngle と getAngle の角度は Stage の setUpwardVec メソッドで指定されたものを基準とする。図で表すと Graph 2.2 の様になる。UpwardVec を Z'として、z 軸方向（上方向）を z'にとなるように回転させ、それと同じように y を y'、x を x'作成する。そしてそれに対する角度 Φ、θ を得る。ただし Φ は latitude（緯度）、θ は longitude（経度）である。第三の回転軸として向いている方向周りの回転があるがこれはまだ実装されていない。



Graph 2.2

・プログラマが作成するべき部分を明確すること

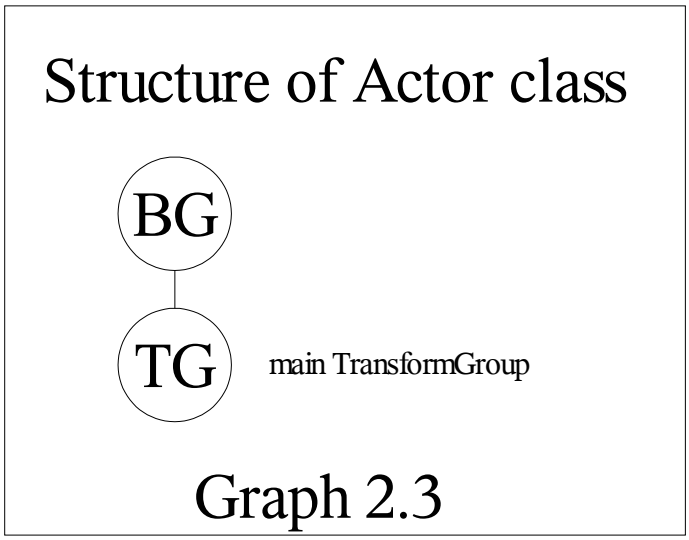
簡単に説明すると三次元オブジェクトを追加する際にプログラマは Actor の getTransformGroup メソッドで返される TransformGroup にオブジェクトを追加すればよい。

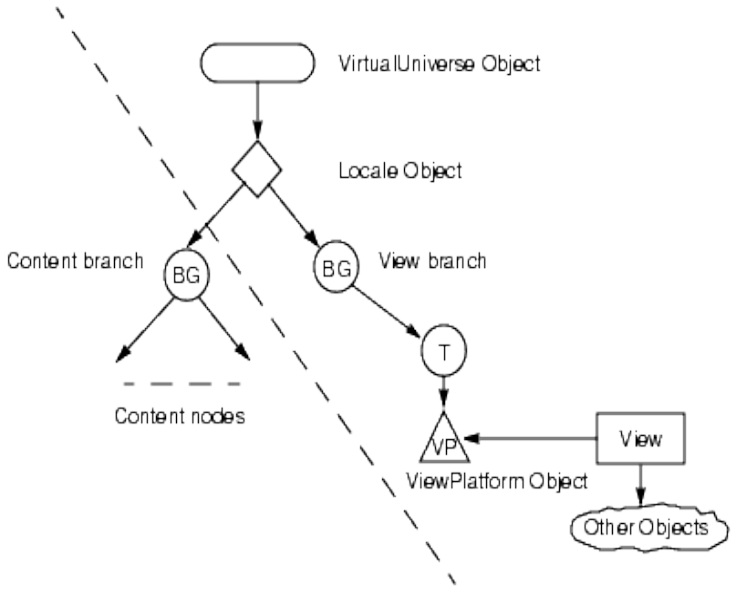
Graph 2.3 に Actor の class が持つ BranchGroup、TransformGroup を示す。この図の通り Actor の class は TransformGroup を１つ持っている。プログラマは getTransformGroup によって TransformGroup を取得しそれに自分が使用したい三次元オブジェクトを追加すれば、後は setPosition や getPosition、setAngle,getAngle メソッドを使用するだけで三次元オブジェクトを動かすことができる。

なお、実装されているメソッド以外を用いて TransformGroup を変更した場合は位置情報なら rehashPosition()、角度情報なら rehashAngle()メソッドを実行してクラスの内部情報に反映する必要がある。

2.4:クラス CameraMan

カメラはその Stage を移すという役割を持っている。このため、グラフィックやサウンド、テキストを出力する先を独自の設定する必要がある。また、次に The Java3D API Specification の図 2.2 を示します。これは Java3D の基本的な仕様を表した物です。カメラは VeiwPlatformObject を持つ TransformGroup 以下の構造と同じ物を持つ必要があるので、StageCameraMan はこの BranchGroup と同じ物を持つために Actor を継承した別クラスとなっています。簡単に言うとクラス CameraMan は Graph 2.3 の sub TransformGroup に ViewPratform Object を持っていると言うことになります。





2.5:クラス Acting

このクラスは Actor クラスが使用する Acting の雛形である。すべての Acting はこのクラスを継承して作成される。以下では extends された Acting のクラスをしばしば「Acting のタイプ」と表記することがある。Acting クラスを Acting クラスの重要なメソッドを次に示す。

protected void setIsActive(boolean arg)
　この Acting が active かどうかを設定する。サブクラスからのみ呼び出し可能。
abstract public boolean doActing()
　この Acting の実行を行うメソッドである。
abstract public boolean setCommand(DataPack datapack)
　datapack を解釈してこの Acting の設定を行う。
public void getIsActive()
　この Acting が active かどうかを返す。true なら active である。
public int getIngerruptedFlag()
　割り込みのフラグを返す。

public Actor getTargetActor()
　このActingの対象となるActorを返す。
public void setInterruptedFlag(int signal)
　割り込みのフラグを返す。

上を見るとabstractがついたメソッドがあるのに気づくと思います。これはActingクラスを継承して新しいActingクラスを作るときに必ず指定しなくてはからです。setCommandメソッドはコマンド（Chapter 3で述べる）からActingの設定を行うための物で、doActingメソッドは実際に行う動作である。

2.6:ActingListクラス

ActingListクラスはActingクラスを継承した物である。このクラスは内部にListを持っており、他のActingを中に格納することができます。Actingとしての性質は、格納されているactiveなActingの中で最も一番最後に格納された物、つまり実行中のActingと同じ性質を持つ。たとえばActingListのdoActingメソッドは、ActingListの実行中のActingを実行するので、Chapter 1で述べた「ActingListを実行する」という動作をします。

ActingListsクラスのdoActing()、setInterruptedFlag(int signal)、getStatus()、getName()メソッドはすべてActingListで実行中のActingに対する物である。詳細はClass Specification参照。

2.7:大まかな制作手順

ここで大まかなStage3D APIを使用してのシーンの作成手順を説明する。作成手順は大きく３つの手順に分かれる。ここでは例として「台所にいる母親のところに少年が帰ってきておやつをねだる」、というシーンの作成手順を説明する。

・Step 1：ActorとActingクラスの作成

ActorとActingクラスを継承して必要なActorとActingクラスを作成する。ここではActingクラスを継承したActWalkクラス作成します。そして、Actorクラスを継承した、少年を表すBoyクラスと母親を表すMotherクラスを作成して、"walt"と言う名前でBoyクラスとMotherクラスに登録します。これはBoyクラスとMotherクラスのコンストラクタに

```
setActing(“walk”,ActWalk.class);
```

を追加すればよい。

・Step 2：Stageの作成とカメラ

まず、Stageクラスを継承した新しいクラスを作成する。ここではそのクラスの名前を"Kitchen”とする。そして台所の三次元オブジェクトを作成する。これはStageクラスがJava3DのLocateクラスを継承しているので、Java3Dで行うように行えばよい。

次に使用するActorとActingを追加する。この例ではBoyクラスとMotherクラスをStageに登録する。これはKitchenのコンストラクタに

```
setActor("Boy",Boy.class);
setActor("Mother",Mother.class);
```

を追加すればよい。

そして、StageCameraMan（もしくは継承したクラス）のインスタンスを作成して、Kitchenに設定する。setCameraManメソッドを使用する。

・Step 3

Step2までによってStageの準備は整いました。次に行うことは「Actorの追加」と「Actorの行うActingの指示」です。たとえばActorを二人追加させたい

場合は以下のような二つの Acting が必要になります。

・StageManager に Boy クラスを「boy」と言う名前で Stage に追加する。

・StageManager に Mother クラスを「Mother」と言う名前で Stage に追加する。

これは Step 2 で登録された"Boy"で登録されたクラスの"boy"インスタンスを作成と"Mother"で登録されたクラスの"mother"を作成することを意味します。

そして、boy を点(-1,1.2,2)まで速度 1 で歩かせたいとすると、

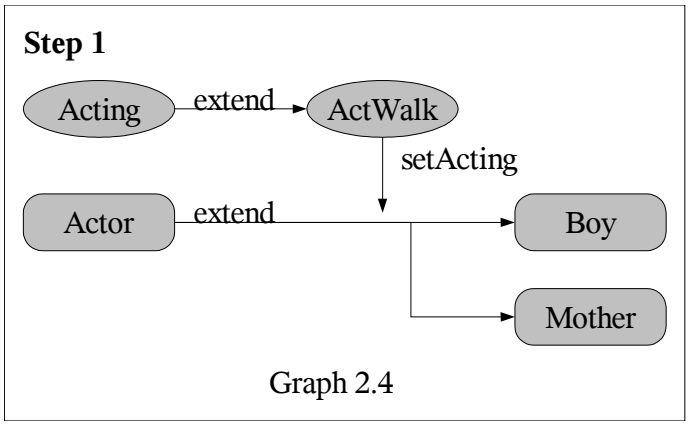
・StageManager に boy を点(-1,1.2,2)まで速度 1 で歩かせるという命令を出させる。

という Acting を StageManager に渡すか、直接 boy に

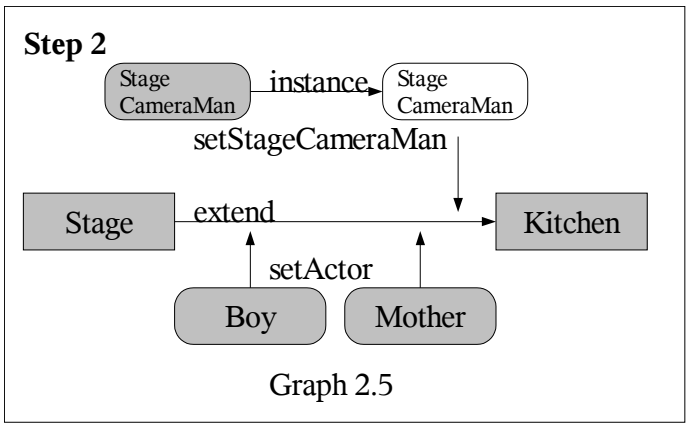
・点(-1,1.2,2)まで速度 1 で歩かせるという Acting

を渡せばいいわけです。

Step 1 と Step 2 を図にすると Graph 2.4 と Graph2.5 になります。この図の灰色はクラスを、白はクラスのインスタンスを意味します。矢印は extend は継承したクラス、instance はインスタンスの作成、それ以外はそれぞれのメソッドを意味します。



Graph 2.4



Graph 2.5

3：Acting とコマンド

3.1:Acting の生成方法

Acting は必要な情報を持った DataPack から生成されます。この DataPack を「コマンド」と呼ぶようにします。コマンドは Stage クラスの execCommand(DataPack command)もしくは Actor クラスの interpret(DataPack command)メソッドの引数として渡されると、メソッド内で Acting に変換され Actor の Queue に追加されます。

3.2:Acting のタイプ

第１章で説明した Acting のタイプは次のようになります。

1) inactive で実行されない物
2) active で実行されるとターンを終了して、有限回実行されると inactive なるもの。
3) active で実行されてもターンを終了せず、有限回実行されると inactive なるもの。
4) active で実行されるとターンを終了して、無限回実行しても inactive にならないもの。
5) active で実行されてもターンを終了せず、無限回実行しても inactive にならないもの。

次の節で説明する物のタイプはこれを指す物とします。

3.3:標準で使用可能な Acting

標準で使用可能な Acting は以下の様になる。これらはすべて Acting クラスを継承した物で Actor クラスで標準で登録されていて使用可能である。なおタイプが複数ある物は使い方によって分かれるものである。

＜クラス名＞	＜名前＞	＜タイプ＞
ActAddActor	add	3
ActEnQueue	queue	3
ActInterrupt	interrupt	3
ActMove	move	2,4
ActSay	say	2
ActSetAngle	setAngle	3
ActSetPosition	setPosition	3
ActThrowEvent	throwEvent	3
ActTurn	turn	2
ActWait	wait	2,4

3.4:コマンドの作り方

コマンドは階層が２以上の DataPack により作られる。コマンドはコマンド名とオプションの二つに分けられる。コマンド名は 3.3 でいう「add」や「move」という Actor に対して登録された Acting の名前である。オプションはその Acting を設定するのに必要な情報である。

DatPack の基本的な構造は次のようになる。第一階層の String クラス配列の第一要素に登録された Acting の名前となり、第一階層の DataPack クラス配列がその Acting のオプションとなる。ただしオプションに DataPack クラス配列を使用しているが標準の Acting は配列の第一要素しか扱っていない。

```
DataPack

  String[0] = Acting Name
  DataPack[] = Option
```

3.5:標準 Acting のオプション

以下は Stage3D API パッケージに含まれる Acting のオプションである。なお表中「Mode Select」とは、その部分の値によって動作が切り替わることを意味している。

Class name	ActTurn			
Default name	turn			
Mode select	intData[0]			
指定された Actor の方向を向く				
intData	0	ロックの有無		
doubleData	速さ			
StringData	振り向く対象			
指定された座標の方向を向く				
intData	1	ロックの有無		
doubleData	x 座標	y 座標	z 座標	速さ
StringData				
指定されたベクトルの方向を向く				
intData	2			
doubleData	x 方向	y 方向	z 方向	速さ
StringData				
指定された角度の方向を向く				
intData	3			
doubleData	緯度	経度	速さ	
StringData				
指定された経度の方向を向く				
intData	4			
doubleData	経度	速さ		
StringData				
指定された緯度の方向を向く				
intData	5			
doubleData	緯度	速さ		
StringData				

Class name	ActSetAngle		
Default name	setAngle		
Mode select	intData[0]		
指定された Actor の方向を向く			
intData	0		
doubleData			
StringData	振り向く対象		
指定された座標の方向を向く			
intData	1		
doubleData	x 座標	y 座標	z 座標
StringData			
指定されたベクトルの方向を向く			
intData	2		
doubleData	x 方向	y 方向	z 方向
StringData			
指定された角度の方向を向く			
intData	3		
doubleData	緯度	経度	
StringData			
指定された経度の方向を向く			
intData	4		
doubleData	経度		
StringData			
指定された緯度の方向を向く			
intData	5		
doubleData	緯度		
StringData			

Class name	ActEnQueue
Default name	queue
Mode select	none
指定された Actor に Acting を追加する	
intData	キュー
doubleData	
StringData	対象 Actor
DataPack	Acting

Class name	ActSetPosition		
Default name	setPosition		
Mode select	none		
指定された位置に Actor を配置する			
intData			
doubleData	x 座標	y 座標	z 座標
StringData			

Class name	ActWait	
Default name	wait	
Mode select	intData[0]	
指定された時間停止する		
intData	0	停止時間
doubleData		
StringData		
指定された Actor のキューが空になるまで停止する		
intData	1	キュー
doubleData		
StringData	対象 Actor	
指定された Actor の指定された Acting が実行中の間停止する		
intData	2	
doubleData		
StringData	対象 Actor	対象 Acting

表 3.1:Acting とそのオプション

Class name	ActAddActor
Default name	add
Mode select	none
Actor に名前を付けて Stage に追加する	
intData	
doubleData	
StringData	Actor Type　Actor Name
DataPack	Option

Class name	ActSay		
Default name	say		
Mode select	intData[2]		
グループに対して発言する（未実装）			
intData	発言時間	ボリューム	0
doubleData			
StringData	対象 Group	対象 Actor	発言
Actor に対して発言する			
intData	発言時間	ボリューム	1
doubleData			
StringData	対象 Actor	発言	
発言する			
intData	発言時間	ボリューム	2
doubleData			
StringData	発言		

Class name	ActCamSetCamTargetActor
Default name	setCamTargetActor
Mode select	none
SimpleCameraMan に対して撮影対象となる Actor 設定する	
intData	
doubleData	
StringData	対象 Actor

Class name	ActMove			
Default name	move			
Mode select	intData[0]			
指定された Actor に近づくように移動				
intData	0	ロックの有無		
doubleData	速さ	停止距離		
StringData	対象 Actor			
指定された座標まで移動				
intData	1			
doubleData	x 座標	y 座標	z 座標	速さ
StringData				
指定されたベクトルの方向へ移動				
intData	2	移動する時間		
doubleData	x 方向	y 方向	z 方向	速さ
StringData				

Class name	ActSetCamTargetPoint
Default name	setCamTargetPoint
Mode select	none
SimpleCameraMan に対してカメラを向ける座標を決定する。対象 Actor が設定されている場合はそこからの座標とする。	
intData	
doubleData	x 座標　y 座標　z 座標
StringData	

Class name	ActSetCamTargetLock
Default name	setCamTargetLock
Mode select	none
SimpleCameraMan が撮影点を追い続けるかを設定する。ロックは値が 0 以外の時に行う	
intData	ロック
doubleData	
StringData	

Class name	ActCamSetCamViewActor
Default name	setCamViewActor
Mode select	none
SimpleCameraMan に対して撮影視点となる Actor を設定する	
intData	
doubleData	
StringData	対象 Actor

Class name	ActSetCamViewPoint
Default name	setCamViewPoint
Mode select	none
SimpleCameraMan に対してカメラ位置の座標を決定する。対象 Actor が設定されている場合はそこからの座標とする。	
intData	
doubleData	x 座標　y 座標　z 座標
StringData	

Class name	ActSetCamViewLock
Default name	setCamViewLock
Mode select	none
SimpleCameraMan が撮影視点を追い続けるかを設定する。ロックは値が 0 以外の時に行う	
intData	ロック
doubleData	
StringData	

Class name	ActInterrupt		
Default name	interrupt		
Mode select	intData[0]		
指定された Actor の Acting に対して割り込む			
intData	0	シグナル	
doubleData			
StringData	対象 Actor	対象 Acting	
指定された Actor のキューに対して割り込む			
intData	1	キュー	シグナル
doubleData			
StringData	対象 Actor		

表 3.2:Acting とそのオプション

3.6:コマンドの例

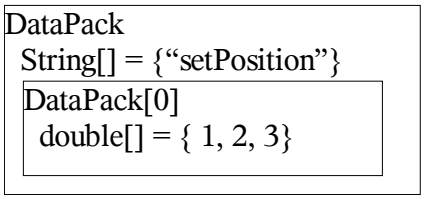
次に簡単なコマンドの例を示す。

例１）Actor を地点{1,2,3}に移動させる。

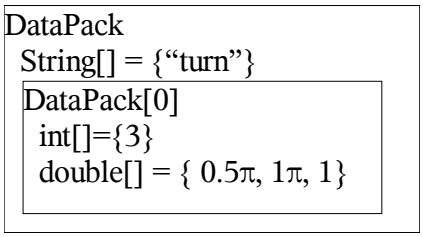
例２）Actor を速さ１回転毎秒で、角度{0.5π, 1π}に振り向かせる。

例３）例２によって生成される Acting を「boy」という Actor のキューに追加する。

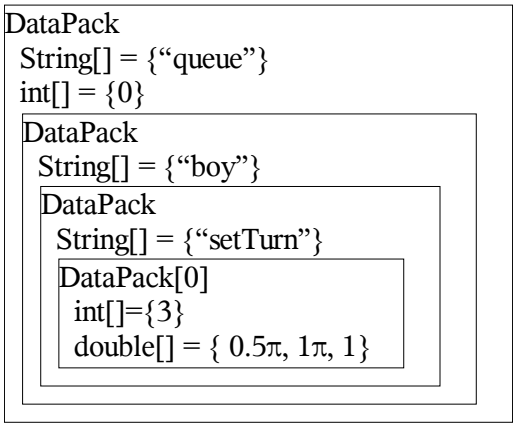
例 1



例 2



例 3



3.7:コマンドの作成の方法

コマンドは DataPack クラスを利用しているため、テキストで書かれた命令のようにプログラム外からコマンドを入力するために少し工夫が必要である。以下ではいくつかの手段を示唆しておく。

- あらかじめ必要なコマンドをプログラム内で作成しておいて、それを利用する。
- DataPack は XML 形式に書き出し・読み込みできるので XML にコマンドを書き出しておく。
- テキスト形式で書かれた命令をコマンドへ変換するプログラムを作っておく。

特に３つ目の方法はサンプルプログラムの StringParser クラスが行っているので興味があるなら目を通して貰いたい。

４：Actor と Acting の作成方法

4.1:Actor の作成方法

Acting クラスは Actor の動作のためのクラスである。よって基本的に Actor クラスに動作についての機能を持たせる必要がない。Actor クラスには三次元オブジェクトの設定と使用する Acting の登録を行えばよい。つまり、Actor クラスを継承して、コンストラクタに３次元オブジェクトの設定と Acting の登録を書けばよい。

たとえば Door クラスのコンストラクタにおいて、３行目の setActing は ActDoorOpen という Acting クラスの派生クラスを"open" という名前で登録している。４行目は TransformGroup を取得して、それに createDoor メソッドによって作成される Door の三次元オブジェクトを追加している。

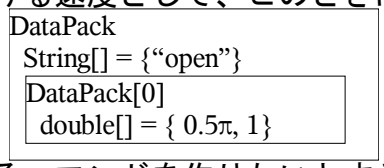
```
public class Door extends Actor
{
   public double openAngle=0;
   public Door()
   {
      setActing("open",ActDoorOpen.class);
      getTransformGroup().addChild(createDoor(0.05, 2.0, 1.0));
   }
   public void setOpenAngle(double angle)
   {
      Transform3D tg = new Transform3D();
      getTransformGroup().getTransform(tg);
      tg.setRotation(new AxisAngle4d(0d,1d,0d,angle));
      getTransformGroup().setTransform(tg);
      openAngle=angle;
   }
   public double getOpenAngle()
   {
      return openAngle;
   }
   private BranchGroup createDoor(double xLeng,double yLeng,double zLeng)
   {
      ????
   }
```

4.2:Acting の作成方法

Acting は Actor の動作を表した物である。Acting を作成するには setCommand メソッドと doActing メソッドの二つのメソッドをオーバーライドする必要がある。setCommand はコマンドのオプション部分が

ここでは上の Door クラス用のドアを開けるという動作の ActDoorOpen という Acting を例にとって説明する。

まずコマンドに付いて考える。Acting の名前を Actor に登録するときに"open"で登録するとする。またオプションの double 型配列の第一要素がドアの開き角、第二要素が開ける速度として、このときに「コマンド」が



となるコマンドを作りたいとする。このソースコードは次のようになる。

```
1)public class ActDoorOpen extends Acting
2){
3)double Speed,rad;
4)   public boolean setCommand(DataPack[] datapacks)
5)   {
6)      setIsActive(false);
7)      if(datapacks.length<1) return false;
8)      DataPack datapack = datapacks[0];
9)      double[] dd = datapack.getDoubleData();
10)     rad = dd[0];
11)     Speed = dd[1]*((double)(getStage().getClockTime())/1000d);
12)     setIsActive(true);
13)     return true;
14)  }
15)  public boolean doActing()
16)  {
17)     if (getInterruptedFlag()!=0){setIsActive(false); return false; }
18)     //？？？？？？？？？？？？？
19)     double radt = ((Door)getTargetActor()).getOpenAngle();
20)     if (rad > radt) radt+=Speed;
21)     else radt-=Speed;
22)     //？？？？？？？？
23)     if (Math.abs(rad-radt)<(Speed/2))
24)     {
25)        ((Door)getTargetActor()).setOpenAngle(rad);
26)        this.setIsActive(false);
27)        return true;
28)     }
29)     ((Door)getTargetActor()).setOpenAngle(radt);
30)     return true;
31)    }
32)}
```

4.3:特殊な例

ここで紹介したActDoorOpenクラスのActingもそうだが、メソッド内にDoorクラスへキャストしてクラス特有のメソッドを使っている。故にこのActingを他のActorの

クラスに使用することはできない。逆に Actor と Acting を専用設計で作成すれば動作の自由度を格段に上げることができる。

4.4:ノウハウ

・Acting を使わない Actor を作りたい

Acting は Actor の doActing()メソッドから呼び出される。Acting を使用しない場合はこの Actor の doActing()メソッドをオーバーライドすればよい。また、動作を追加したい場合はオーバーライドして super.doActing()を呼び出せばよい。

・Actor から他の Actor の状態を確認したい

Actor クラスの getStage()メソッドによって Stage を取得した後に、getActor (String actorname)メソッドを呼び出せば他の Actor を取得することができる。

５：Stage3D.utilsパッケージ

5.1:SimpleCameraMan

SimpleCameraManクラスはCameraManクラスの派生クラスであり、独自Actingをもつことでカメラワークの自由度を高めた物である。基本的に基本Actngは使用不可能になっている。

このクラスはCamTargetActor、CamTargetPoint、CamTargetLock、CamViewActor、CamViewPoint、CamViewLockの６つのフィールドを持っている。CamTargetActor、CamTargetPoint、CamTargetLockの３つのフィールドはカメラの向きを、CamViewActor、CamViewPoint、CamViewLockの３つのフィールドはカメラ位置を決定するのに使用される。

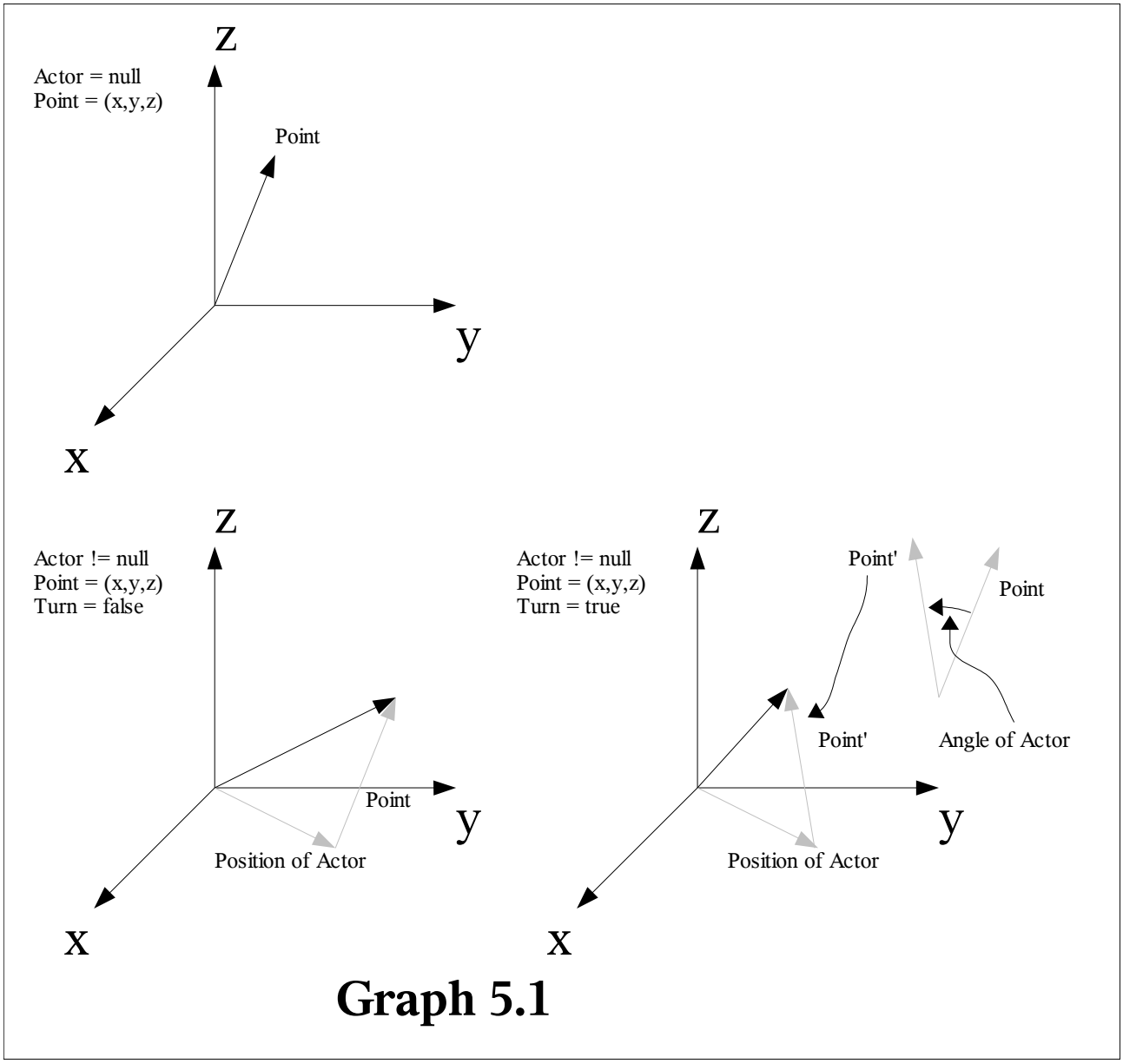
◇カメラ位置の決定方法は３通りある
- ◆CamViewPointがnullのとき
  カメラの位置は変更されない
- ◆CamViewPointがnullでないとき
  - ◎CamViewActorが存在しているとき
    - ●CamViewLockがfalseのとき
      カメラはCamViewPoitとCamViewActor.getPosition()の和の位置に設定される。
    - ●CamViewLockがtrueのとき
      カメラはCamViewPointとCamViewActor.getPosition()をCamViewActor.getRotationにて回転させたベクトルの和の位置に設定される。
  - ◎CamViewActorがnullのとき
    カメラはCamViewPointが指す点の位置に設定される

◇カメラの向きの決定方法は３通りある
- ◆CamTargetPointがnullのとき
  カメラの向きは変更されない
- ◆CamTargetPointがnullでないとき
  - ◎CamTargetActorが存在しているとき
    - ●CamTargetLockがfalseのとき
      カメラはCamTargetPointとCamTargetActor.getPosition()の和の位置の方向を向く
    - ●CamTargetLockがtrueのとき
      カメラはCamTargetPoitとCamTargetActor.getPosition()をCamViewActor.getRotationにて回転させたベクトルの和の位置の方向を向く
  - ◎CamTargetActorがnullのとき
    カメラはCamViewPointが指す点の方向を向く

カメラの位置と方向はオーバーライドされたdoActingメソッド内によって更新される。この処理はActingのActMove、ActTurn、ActSetDirection、ActSetPositionと競合するため、これらのActingはSimpleCameraManでは使用できなくなっている。代わりにActCamSetCamTargetActor、ActCamSetCamTargetPoint、ActCamSetCamTargetLock、ActCamSetCamViewActor、ActCamSetCamViewPoint、ActCamSetCamViewLockが使用可能になっている。これらのActingのコマンドは３章で示したとおりである。

カメラの位置と方向はカメラ位置の点と撮影点は二つの点を表すベクトルにより決定される。それぞれの点はActor、Point、Lockの３つの要素からなる。その三つの要素から生成される位置ベクトルはそれぞれの状態によってGraph 5.1によって分類される。Graph 5.1でActor、Point、Lockはカメラ位置の点についてはActCamSetCamViewActor、ActCamSetCamViewPoint、ActCamSetCamViewLockによって設定され、撮影点についてはActCamSetCamTargetActor、ActCamSetCamTargetPoint、ActCamSetCamTargetLockによって設定される。そして

カメラ位置の点と撮影点が設定されたのち、カメラ位置の点にカメラを位置を設定し、カメラの向きをカメラの位置から撮影点の方向を向くように設定する。



Graph 5.1